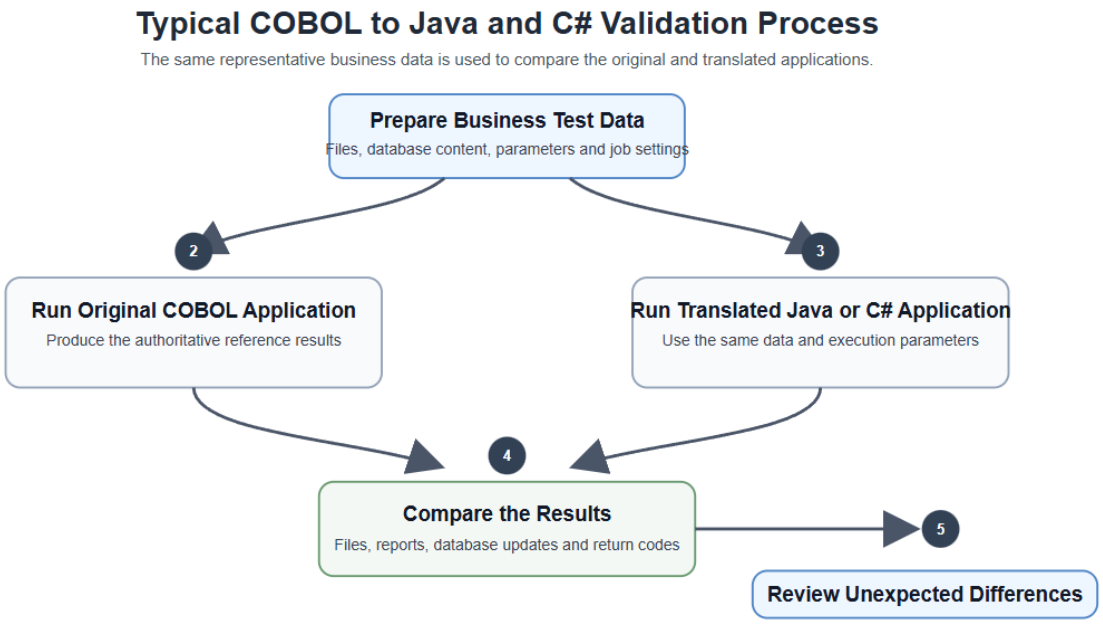
COBOL to Java and C# Testing Strategy

One of the most common concerns raised during a COBOL migration project is the amount of testing required. Many organizations assume that migrating to Java or C# means creating an entirely new suite of functional tests. In practice, this is rarely necessary. The original COBOL application already defines the required behavior, making it the reference implementation against which the translated application is validated.

1 COBOL Migration Testing Summary

- The original COBOL application serves as the reference implementation for all testing.
- Existing business test data and operational scenarios can normally be reused. There is usually no need to create an entirely new suite of functional tests.
- The primary testing activity is validating that the translated Java or C# application produces the same outputs, reports and database updates as the original COBOL system.
- SoftwareMining provides deterministic translation, producing consistent Java and C# architectures that simplify code reviews, regression testing and long-term maintenance.
- Utilities for EBCDIC conversion, VSAM migration, translated JCL execution and output comparison further simplify the migration testing process.



Key Takeaway: A COBOL to Java or C# migration does not normally require a new functional test suite. The original COBOL application provides the reference implementation, allowing the translated application to be validated by comparing the results produced from the same business data.

For COBOL batch applications, testing is normally performed end-to-end using representative business data. The original COBOL program and the translated Java or C# application are executed with the same input files, database content and execution parameters. Where required, SoftwareMining utilities convert EBCDIC files to ASCII or Unicode and execute translated JCL through equivalent Windows or Linux scripts.

The results produced by both applications are then compared, including output files, reports, database updates, return codes and exception behavior. Because the original COBOL application acts as the functional specification, successful comparison provides evidence that the translated application preserves the required business behavior.

2 How Deterministic Translation Reduces Testing Effort

SoftwareMining uses a deterministic translation engine that applies the same translation rules to every COBOL program. The same COBOL construct always generates the same Java or C# implementation pattern, producing a consistent architecture across the entire application. This consistency simplifies code reviews, testing and long-term maintenance because developers encounter familiar programming patterns throughout the migrated system.

Common COBOL infrastructure is implemented within well-tested SoftwareMining runtime libraries rather than repeatedly generating identical code in every translated program. File handling, database access, transaction management and data conversion are therefore implemented once and reused throughout the application. For example, every COBOL OPEN, READ, WRITE and CLOSE operation is translated into the same runtime library calls throughout the application. This reduces generated code, improves maintainability and allows testing to concentrate on the translated business logic.

For a technical evaluation comparing AI-assisted translation with SoftwareMining's deterministic translation approach, please see Claude Code Technical Evaluation for Enterprise COBOL Modernization.

3 Typical COBOL to Java and C# Validation Process

1. Prepare representative business data and expected operational scenarios.
2. Execute the original COBOL application to produce the reference results.
3. Execute the translated Java or C# application using the same inputs.
4. Compare output files, reports, database updates and return codes.
5. Investigate and resolve any unexpected differences.

Additional testing may still be required for the target environment, including performance, security, deployment, integration and operational testing. These activities are separate from functional-equivalence testing and would normally be required for any significant platform change.

Once the translated application has been accepted, subsequent enhancements are normally made directly to the Java or C# source code. Unlike generative AI approaches, where regenerated code may differ between runs or after prompt changes, the SoftwareMining translator is deterministic. The same COBOL source always produces the same Java or C#

implementation. This repeatability reduces validation effort because validated translation patterns remain consistent throughout the application and across subsequent translations.

4 Conclusion

Functional-equivalence testing provides a practical and auditable way to validate a COBOL migration. By executing the original and translated applications with the same representative business data, organizations can compare observable results and investigate any differences before deployment. SoftwareMining's deterministic translation patterns and supporting conversion and comparison utilities make this process consistent across large Java and C# migration projects.

5 COBOL Migration Testing Checklist

- Are we using the existing COBOL application as the reference for expected behavior?
- Can existing business scenarios and test data be reused?
- Will both applications be run with the same files, database content and execution parameters?
- Are output files, reports, database updates and return codes compared?
- Are exception paths and error conditions included where they affect business behavior?
- Is the comparison process repeatable and auditable?
- Are platform services such as CICS, IMS, DB2, VSAM and JCL covered by the migration approach?
- Is performance and deployment testing treated separately from functional-equivalence testing?
- Can the migrated application be regenerated consistently after COBOL source changes?

See: Runtime Performance Comparisons